Grid4All Security User's Manual, Release 0.6

by Leif Lindbäck and Vladimir Vlassov
Royal Institute of Technology (KTH), Stockholm, Sweden
Email {leifl, vladv}@kth.se

FP6 Project Grid4All (IST-2006-034567)

Contents

Abstract 3

1 Grid4All Security Architecture 3
1.1 Policy Enforcement Point (PEP) 4
1.2 Policy Decision Point (PDP) 4
1.3 Policy Information Point (PIP) 4
1.4 Policy Administration Point (PAP) 4
1.5 Policy Repository (PR) 5

2 Installation 5

3 Running Security Components 5
3.1 Running PDP 5
3.2 Running the PEP Demo (a PDP Client) 6
3.3 Running PAP 6
3.4 Running the PAP client 8
3.5 Running VOMS 8

4 Programming Security Components 8
4.1 Programming Policy Decision Points (PDP) 8
4.1.1 Interaction with PDP over a TCP Connection 9
4.1.2 Interaction with PDP through Standard Input and Output 9
4.1.3 Interaction with PDP Using RMI 9
4.1.4 Interaction with PDP by Local Method Invocation 10
4.2 Programming Policy Enforcement Points (PEP) 11
4.2.1 The class `CachingPEP` 11
4.2.1.1 Cache Invalidation 11
4.2.1.2 Communication with PDP 11
4.2.2 Interaction with PEP 12
4.2.2.1 Interaction with PEP by Local Method Invocation 12
4.2.2.2 Interaction with PEP using Streams 12
4.3 Programming Policy Administration Points (PAP) 14
4.3.1 Interaction with PAP over a TCP Connection 14
4.3.2 Interaction with PAP Using RMI 14
4.3.3 Interaction with PAP by Local Method Invocation 15

5 Limitations 15

6 Known Bugs 15

Abstract

This User's Manual is for developers who intend to use Grid4All security API for authentication and authorization in their distributed applications and systems.

The Grid4All security API allows building a policy-based security infrastructure where policies are expressed in XACML (eXtensible Access Control Markup Language). The Grid4All security API (classes and interfaces) has been developed using Sun's XACML implementation (http://sunxacml.sourceforge.net/).

The Grid4All security distribution (including source and documentation) is available at http://www.isk.kth.se/~leifl/vofs/

This User's Manual is structured as following. First, it describes the Grid4All security architecture; next it gives installation instructions; followed by start-up instructions and user commands; and finally, it presents a programmer's guide that explains how to use Grid4All security API for programming a policy-based security infrastructure in distributed applications and systems.

1 Grid4All Security Architecture

The Grid4All security includes the following security components (Figure 1).

- *Policy Enforcement Point* (PEP) enforces VO policies;
- *Policy Decision Point* (PDP) makes authorization decisions based on VO policies;
- *Policy Information Point* (PIP) collects user security-related data;
- *Policy Administration Point* (PAP) is used to administrate (create, update, delete) policies;
- *Policy Repository* is a persistent store for policies;
- *VO Membership Service* (VOMS) maintains VO membership.

PEP is the only component that resides on the client side and may need to be customized or re-implemented by the developer. Other components remain unchanged across different application domains and use-cases. The components are explained in more detail below.

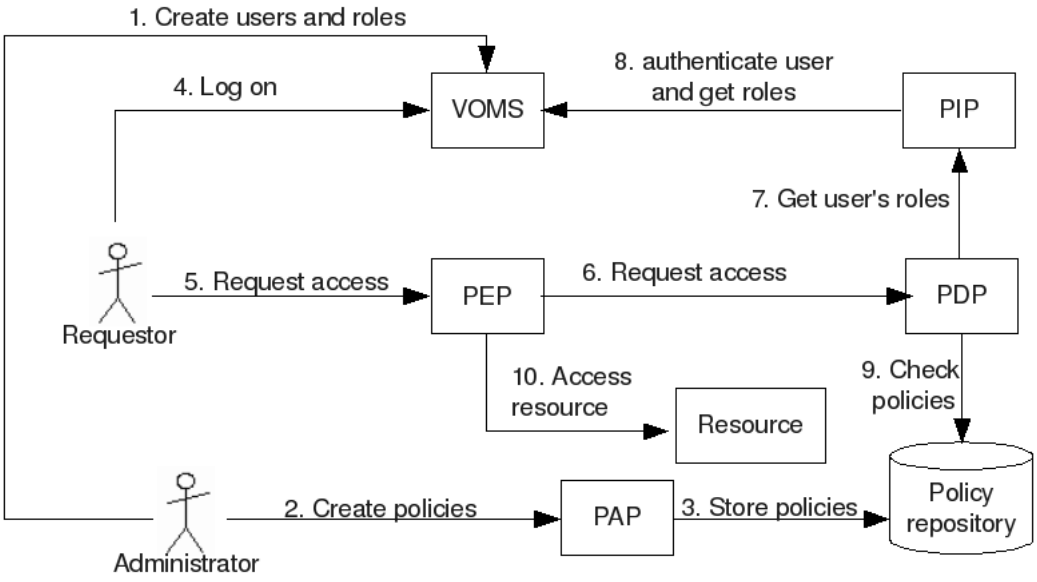

Figure 1. Security architecture

1.1 Policy Enforcement Point (PEP)

PEP protects a resource. It is called whenever access rights shall be checked. An authorization request to PEP includes the following three parameters.

- *Subject* which is the single sign-on identifier of the user accessing the resource. This identifier was returned by the VOMS when the user signed on.
- *Action*, which specifies a name of an action to be performed on the resource.
- *Resource* which is the target resource identifier in the form of a path starting with `@` (e.g. `@/se/kth/ict`).

PEP sends the authorization request to PDP and upon receiving an authorization response from PDP, it enforces the authorization decision. Note that the PEP classes provided in the Grid4All API, do not enforce the authorization decisions but only cache the decisions communicate them to a component requesting the access check. The Grid4All Security API includes a sample PEP class that only caches authorization decisions (see 4.2). This class can be used as a template to develop a PEP enforcing authorization decisions. Caching of PDP responses at PEPs reduces security overhead. The developer can use PEP either with or without cache for PDP responses. The PEP class provided in Grid4All security API supports caching of PDP responses

Note that any object sending authorization requests to PDP and receiving an authorization response could be considered as PEP.

1.2 Policy Decision Point (PDP)

The PDP evaluates requests in the context of a security policy. When the PDP receives an authorization request from PEP, it retrieves the appropriate policy from the Policy Repository or from its policy cache maintained in memory, evaluates the request, makes the authorization decision, and returns it to the requesting PEP as an authorization response. The response includes one of the following possible results:

- *Permit*, this means that access is granted.
- *Deny*, this means that access is rejected.
- *NotApplicable*, this means that there was no matching policy.
- *Indeterminate*, this means that no decision could be taken. For example there might be several contradicting policies.
- *Error*, which means that policies could not be checked because of some exception, for example the communication link might be broken

The PDP maintains a cache of policies which are loaded from the Policy Repository.

1.3 Policy Information Point (PIP)

PIP is responsible for authenticating users and for retrieving their roles from the VOMS. PIP caches VOMS answers.

1.4 Policy Administration Point (PAP)

PAP is used to administer (list or change) policies stored in the Policy Repository (PR). Before accessing the repository, PAP calls its own PEP to check if the policies in PR grant the caller the right to do the requested operation. The provided implementation of PDP assumes that XACML policies are stored as XML files.

1.5 Policy Repository (PR)

The policy repository is a persistent store for policies in the eXtensible Access Control Markup Language (XACML)[1].

2 Installation

1. Download the Grid4All security distribution from `http://www.isk.kth.se/~leifl/vofs/`
2. Unpack the content of the archive
3. Check and edit entries (e.g. host names, port numbers, paths, etc.) in two configuration files, `etc/config/grid4all.secserv.config` and `etc/config/grid4all.pep.config` for PDP/PAP and PEP, respectively. The entries are documented in the files. When running a program, e.g. PDP, PAP or PEP, the paths to configuration files should be either on the class path or specified with the `-fs` option for the PDP/PAP configuration file (`grid4all.secserv.config`) and `-fp` for the PEP configuration file (`grid4all.pep.config`).
4. Make sure that the `policyPath` entry in the `grid4all.secserv.config` configuration file points to a directory that contains the file `etc/policy/default-admin.xml`, otherwise no user will have the right to execute any commands on security components. The `default-admin.xml` file grants the role `admin` all rights to all resources protected by Grid4All security.

3 Running Security Components

PDP (Policy Decision Point), PAP (Policy Administration Point) and VOMS (VO Membership Service) are servers. This section explains how to start PDP, PAP and VOMS, and user commands to control them. It also explains how to run the PEP demo and the PAP client. The PEP demo illustrates interaction of PEP with PDP: it allows the user to send commands to PDP over a TCP connection. The PAP client allows the user to send commands to PAP over a TCP connection.

3.1 Running PDP

Start PDP with the `start-pdp` command in the `bin` directory. Communication with PDP can be done through TCP connection, RMI, local method invocation and standard input/output. The RMI and local methods are described in 4.1.34.1.4 respectively. The TCP-based and standard input/output interfaces accept the following commands.

- **`REQUEST <subject> <action> <resource>`**

  Queries the policy repository. The response is one of `Permit`, `Deny`, `NotApplicable` or `Indeterminate` according to the XACML specification. If the request does not have the correct format or some other exception occurs, then the response is `ERROR`. Note that `subject` should be the single sign-on identifier returned by VOMS when the user signed on. PDP will call VOMS to check that the user is signed on and to get the roles of the user. All roles will be evaluated and the result will be `Permit` if any of the roles has the right to perform the specified action. PDP will answer not only to the specified request, but also to other requests with the same `subject` and `resource` but with all existing actions. The response format is

[1] http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf

<answer to specified request**>** **<**nextAction**>** **<**answer to a request with the originally specified subject and resource, but with action replaced by nextAction**>**

The last two items are repeated for each action that exists in the policy repository.

- **REGISTER <port>**

  Registers the given port number and the IP address from which this command is sent, at this PDP to send cache invalidation requests. <port> defines the port number to which the PDP will send cache invalidation requests. This command can only be used from the TCP interface. On success, the command returns REGISTER_OK. When the cache shall be invalidated PDP sends INVALIDATE to the specified port on the IP address from which the REGISTER command was sent. On receiving this command PEP should discard its entire cache. It should not send any answer to PDP. PEP only gets one chance to discard its cache; if this message is lost it will not be retransmitted.

- **UNREGISTER <port>**

  PEP at the address from which the command was sent will stop receiving information about cache invalidations. <port> should be the same as in the REGISTER command. This command can only be used from the TCP interface. On success, the command returns UNREGISTER_OK.

- **RELOAD**

  Causes PDP to discard its policy cache and reload it from the policy repository. The INVALIDATE command (see above) is sent to all registered PEPs. On success, the command returns RELOAD_OK.

- **STOP**

  Terminates PDP. Note that the entire PDP terminates, not only the command interpreter. PDP is shutdown gracefully. All ongoing work will be finished before shutting down. This command can only be used from the standard input interface.

3.2 Running the PEP Demo (a PDP Client)

The PEP demo illustrates interaction of PEP with PDP: it allows the user to send commands to PDP over a TCP connection. To run the PEP demo use the pep-demo command in the bin directory. The demo uses TCP to communicate with PDP that must be started with the start-pdp command described in 3.1. The PEP demo reads the PDP host name and port number from the client configuration file, etc/config/grid4all.pep.config.

3.3 Running PAP

Start PAP with the start-pap command in the bin directory. Note that PAP will call PDP, so you must start that one also. Communication with PAP can be done through TCP connection, RMI and local method invocation. The RMI and local methods are described in 4.3.2 and 4.3.3 respectively. The TCP interface accepts the following commands.

- **<issuer> setacl -dir <directory> -acl <role> <rwilkad> [-clear] [-time <startTime> <endTime>] [-date <startDate> <endDate>]**
  [-dayofweek <startDayOfWeek> <endDayOfWeek>][-negative]

The setacl command sets the access control list (ACL) entries specified with the -acl argument to the ACL of the directory named by the -dir argument for role <role>. On success, the command returns OK otherwise an error message.
The -clear flag is used to remove the ACLs of the <role> on the directory <directory>. In this case, there is no need to specify the permissions (<rwilkad>) since all the permissions will be erased. The -negative flag is used to impose negative ACLs (i.e. "deny" decision) for the specified actions in <rwilkad> for role <role> on the directory <directory>. Note that issuer should be the single sign-on identifier returned by the VOMS when the user signed on. All roles of the user will be evaluated.
The -time, -date and -dayofweek flags makes the policies, whether positive or negative, valid only the specified periods. Periods can not be open; they must always have both start and end. Time format is hh:mm:ss[TZ]. Time zone is specified as +/- offset from UTC, e.g. 00:00:00+02:00. Time periods are allowed to wrap around midnight. Date format is yyyy-mm-dd. Dates must be valid, i.e. November 31 or February 29 of non leap years are not accepted. All years from 0000-9999 are accepted. Day of week format is strings containing day names in English without abbreviations. Day names are not case sensitive. Day of week periods are allowed to wrap, which means that it is not important which day is considered the first of the week. Time zone is not considered when date and day of week rules are evaluated; this means that midnight always occurs when it occurs on the computer evaluating the time period. The -time, -date and -dayofweek flags can be combined in any way and the rule only applies if all of them match.

Example:
```
123456 setacl -dir @/se/kth/ict -acl teacher rwid -time
01:00:00 13:00:00
```
The specified role, teacher, is given the specified permissions, rwid which is short for read, write, insert and delete, on the specified resource, @/se/kth/ict. The permissions are valid from 1 AM to 1 PM.

Privilege Required:
The issuer must have the a (admin) permission.

Answer:
The answer is OK.

- **<issuer> listacl -dir <dir/file path>**

  The listacl command displays the access control list (ACL) of the <role> associated with the specified file or directory. Note that issuer should be the single sign-on identifier returned by VOMS when the user signed on. All roles of the user will be evaluated.

  Example:
  The response of command: 123456 listacl -dir @ might be:
  ```
  ACL for <@ admin>
       Positive Entry: r valid: 01:00:00-02:00:00
       Positive Entry: w valid: always
  ```

```
Negative Entry: a valid: Monday-Tuesday
```

Privilege Required:
The issuer must have the `r` (read) permission on its ACL and the ACL for every directory that precedes it in the pathname. Otherwise PAP returns an error message. Also, it means that you can not see your own ACL on the directory if you don't have `r` (read) permission on the ancestor directories.

Answer:
See *Example* above.

3.4 Running the PAP client

The PAP client allows the user to send commands to PAP over a TCP connection. To run the the PAP client, use the `pap-client` command in the `bin` directory. The PAP client uses TCP to communicate with PAP that must be started with the `start-pap` command described in 3.3. The client reads the PAP host name and port number from the client configuration file, `etc/config/grid4all.pep.config`.

3.5 Running VOMS

1. Initialize VOMS database with the command `bin/init-voms`.
2. Start VOMS with the command `bin/start-voms`. VOMS can now be accessed with a browser at the url `http://<voms host>:8080/voms`.
3. To create users and roles, log in as the default admin user:
   ```
   Username: admin
   Password: g4all
   ```
   It is not necessary to enter anything in the fields `VOFS Host` and `VOFS Port` to be able to create users and roles.
4. The following actions exist in VOMS:
   - `login.jsp` (default) authenticates the user and creates a single sign-on token which is sent to the specified host and port.
   - `GetAllPeerAction.do` returns a list with the host and port of all signed-on users.
   - `CheckIdAction.do?id=<single sign on token>` returns all roles of the user with the specified token. Shows an empty page if there is no such user.

4 Programming Security Components

This section explains how to access Grid4All Security components (PDP, PEP and PAP) programmatically.

4.1 Programming Policy Decision Points (PDP)

PDP is provided as a server application (service) that can be accessed by clients in different ways as explained below. The provided PDP is complete and does not need to be extended in any way. It is defined in the primary class `se.kth.grid4all.security.pdp.PDP` and it is started with the `start-pdp` command in the `bin` directory. When PDP is running,

a client (e.g. PEP) interacts with it by submitting an authorization request and getting in return an authorization response (see Figure 1). The client can communicate with PDP in four different ways: over a TCP socket connection; through the standard input and output streams; using RMI; and by local method invocation on PDP. In the last case, the requesting client must be in the same JVM as the PDP object.

4.1.1 Interaction with PDP over a TCP Connection

The PDP object listens for TCP connections on the port specified in the configuration file, `grid4all.secserv.config`. The accepted commands are specified in 3.2.

4.1.2 Interaction with PDP through Standard Input and Output

Commands can be given through standard input; whereas response can be read from standard output. The accepted commands are specified in 3.1. Note that the command interpreter that reads standard input runs in the same JVM as the PDP itself.

4.1.3 Interaction with PDP Using RMI

The PDP is bound in the RMI registry under the name specified in the configuration file `grid4all.secserv.config`. The remote interface for interaction with PDP is `se.kth.grid4all.security.communication.SecurityServer`. The interface defines the following remote methods.

```
EvaluationResult evaluateRecursively(String subject,
                                     String resource,
                                     String action)
                              throws PolicyException,
                                     RemoteException
```

> Evaluates an authorization request, which includes *subject*, *resource*, and *action*, in the context of security policies, i.e. evaluates whether the given subject has rights to take the given action on the given resource. First checks for a policy for the resource specified by the `resource` parameter. If there is no matching policy, drops the part after the last path delimiter, /, and tries again. This is repeated until a matching policy is found or there is no more path delimiter in resource. The request is evaluated in the context of the found policies.
>
> The method takes the following three parameters. `subject` is the single sign-on identifier of the user wanting to access the resource. The PDP first asks the PIP to call the VOMS to check that this identifier belongs to a signed on user and, if so, gets the roles that this user belongs to. Resolved mapping of identifiers to roles are stored in cache at PIP, where cache entries are valid for the time specified by the value of the key `SSOTimeout` in the configuration file `grid4all.secserv.config`. The parameter `resource` specifies the resource the subject tries to access. The parameter `action` specifies the action the subject wants to take on the resource.
>
> The method returns the PDP's authorization response, which is one of *Permit*, *Deny*, *NotApplicable* or *Indeterminate* (see 1.2). All of the subject's roles are evaluated and the result is *Permit* if any of these roles are permitted to perform the specified action. The returned `EvaluationResult` object contains the PDP's response for the specified `subject`, `action` and `resource`. It also contains the PDP's response to requests for the same `subject` and

`resource`, but with all existing actions. The javadoc for the `EvaluationResult` class describes how to retrieve these responses.

`String reload() throws PolicyException, RemoteException`

Causes the PDP and all registered PEPs to discard their caches. Note that the PDP can only invalidate caches of registered PEPs. PEPs inheriting the `CachingPEP` class are automatically registered at the PDP.

4.1.4 Interaction with PDP by Local Method Invocation

This can be done from an object in the same JVM as the PDP. The following methods exist:

```
public String evaluateSSOIdRecursively(String SSOId,
                                       String resource,
                                       String action)
                                throws PolicyException
```

Behaves the same way as the remote method `evaluateRecursively` (see 4.1.3) except that only the answer to the specified subject, resource and action is returned. There is no prefetching of responses to authorization requests with other actions. The method is defined in the class `se.kth.grid4all.security.pdp.PDP`.

`public void reload() throws PolicyException`

Discards all policy information cached by the PDP, and then loads all policies described in the XACML files stored in the policy directory. The policy directory path is specified in the `grid4all.secserv.config` configuration file. Files in subdirectories of that policy directory will not be read. Calling this method will not affect PEP caches. This method is defined in the class `se.kth.grid4all.security.pdp.PDP`.

4.2 Programming Policy Enforcement Points (PEP)

All requests accessing resources protected by the Grid4All security infrastructure should pass through PEP. PEP should run in the same process (JVM) as the resource it protects. This way, it is impossible to access the resource without being allowed by PEP. For example, a PEP can be placed as Servlet filter when protecting a Web-application. A distributed application may have several PEPs that could interact with the same or different PDPs. One of the issues to be considered when using the security components is an efficient and effective placement of PEPs (and other components) in order to achieve necessary protection of resources while reducing the security overhead. A resource owner can place its own PEP and PDP in addition to those placed by VO administrator.

The PEP implementation provided in the Grid4All security API performs communication with PDP (sending authorization requests and receiving authorization decisions) and caching of decisions. It does not enforce authorization responses received from PDP; it only caches the responses and passes them to clients. The PEP classes (see below) are provided as examples and as utilities for handling response caching and communication with the PDP. The caching functionality is explained in 4.2.1.

The enforcement of security policies, as expressed in authorization responses, must be implemented in the code calling the PDP (or provided PEP).

4.2.1 The class *CachingPEP*

The `se.kth.grid4all.security.pep.CachingPEP` class is a base class implementing a stateful PEP which caches requests (those already sent to the PDP) and the respective responses (received from PDP). Cache entries are valid for the time specified by the value of the key `SSOTimeout` in the `grid4all.pep.config` file. To ease of use, most of the PEP related functionalities are implemented and put in the `CachingPEP` class. Your application-specific PEP would be a subclass of `CachingPEP` implementing the `doEvaluate` abstract method that should return an authorization response (decision) for the given authorization request. In this method, the programmer has the choice between three possible alternatives to get the authorization decision: (1) using cached decision; (2) asking for new decision from PDP; and (3) do some custom evaluation.

When an authorization request is sent to the PDP, it evaluates and returns decisions not only for the specified request, but also for requests with the same subject and resource but with all existing actions. `CachingPEP` will save all these authorization decisions in the cache.

The source code of the `se.kth.grid4all.security.pep.PEP` class, a subclass of `CachingPEP`, is an example that illustrates use of `CachingPEP` functionality.

4.2.1.1 Cache Invalidation

When a `CachingPEP` object is constructed, it registers itself to the PDP specified in the configuration file `grid4all.pep.config`, in order to subscribe for cache invalidation notifications. The syntax of cache registration and unregistration commands can be found in 3.1. The registration is persistent and remains even if the PDP is restarted.

4.2.1.2 Communication with PDP

`CachingPEP` uses TCP for communication with PDP. The end-point (host and port) of the PDP is specified in the `grid4all.pep.config` file. PEP opens a TCP connection to PDP

for each request, receives the response and then closes the connection (see the `callPDP` method in the source code of the `CachingPEP` class).

4.2.2 Interaction with PEP

A client can interact with PEP in two different ways, by local method invocation or through streams.

4.2.2.1 Interaction with PEP by Local Method Invocation

In order to communicate with PEP by local method calls, the calling thread should reside in the same JVM as the PEP. PEP can be created by instantiating either a concrete subclass of `CachingPEP`, or the `se.kth.grid4all.security.pep.PEP` class which contains the minimum functionality needed by such a subclass. The methods that can be invoked on PEP are as follows.

```
public String evaluate(String subject,
                       String action,
                       String resource)
                throws java.io.IOException
```

Handles the cache and evaluates a given authorization request. The actual evaluation is delegated to the `doEvaluate` method. The `evaluate` method is the method that should be called on the concrete PEP subclass when a request is to be evaluated. String match is used for cache lookups. If `subject`, `action` or `resource` differs in any way, then cache entries will not be found. This method is not thread safe. Note that `subject` is the single sign-on identifier returned by VOMS when the user signed on. It is converted to the user's roles as described in 4.1.3

```
public void register(String pdpHost, int pdpPort)
              throws java.io.IOException
```

Makes PEP listen for cache invalidations from the specified PDP. PEP is allowed to register with multiple PDPs. Invalidation from any PDP will cause the entire cache to be discarded.

```
public void unregister(String pdpHost, int pdpPort
                throws java.io.IOException
```

Makes the PEP stop listen for cache invalidations from the specified PDP. Unregistering with a PDP with which we have not registered will not have any effect at all. Note that calling this method is the only way to unregister since PEP registrations are not discarded by PDP restarts.

```
public void reload() throws java.io.IOException
```

Makes PDP discard all policies in its cache and reload them again. Also, all PEPs registered with that PDP will be notified to discard their caches.

In addition, there is a helper method `callPDP` that subclasses of `CachingPEP` can use to call PDP.

4.2.2.2 Interaction with PEP using Streams

The class `se.kth.grid4all.security.pep.StreamReaderPEP` is a concrete subclass of `CachingPEP` that reads PEP commands from an input stream and sends answers

to an output stream. The input and output streams are specified in the constructor when an instance is created. The object starts handling requests when the method `readFromStream` is invoked. There is a command, `PDP_HOME/bin/start-stream-reader-pep`, that starts a `StreamReaderPEP` object (by calling `readFromStream`) which reads commands from the standard input stream and sends answers to the standard output stream. It can handle all commands that the TCP interface of PDP can handle (see 3.1). It can also handle the commands that are accepted by PAP (see 3.2). The host and port of PDP and PAP are specified in the `grid4all.pep.config` file.

4.3 Programming Policy Administration Points (PAP)

PAP is provided as a server application that can be access by clients in different ways as explained below. The provided PAP is complete and does not need to be extended in any way. It is defined in the class `se.kth.grid4all.security.pap.PAP` and is started with the `start-pap` command in the `bin` directory.

A client (e.g. a policy editor or the provided PAP client described in 3.4) interacts with PAP by submitting requests to list or update policies stored in Policy Repository (see Figure 1). The client must have enough rights to issue the policy administration commands (see 3.2).

The client can communicate with PAP in three different ways: over a TCP connection; using RMI; or by local method invocation on PAP. In the last case, the client must be in the same JVM as PAP.

4.3.1 Interaction with PAP over a TCP Connection

PAP listens on the TCP port specified in the config file, `grid4all.secserv.config`. The accepted commands are specified in 3.2.

4.3.2 Interaction with PAP Using RMI

The name that PAP is bound under in the RMI registry is specified in the configuration file `grid4all.secserv.config`. The remote interface is `se.kth.grid4all.security.communication.SecurityServer`. The interface includes the following methods.

```
public void clearACL(String issuer, String resource,
                     String role)
              throws PolicyException, RemoteException
```

Removes an ACL entry from the Policy Repository. There is a one to one mapping between ACL and XACML policy file. So, successful result of this action will be deleting the corresponding file from the repository directory.

```
public void setACL(String issuer, String resource,
                   String role, String permission,
                   boolean positive)
            throws PolicyException, RemoteException
```

Handles the `setacl` (ACL creation) requests. If there exists already a policy for the given `subject`, `resource` pair, then it will be updated; otherwise, a new policy will be generated.

```
public void setACL(String issuer, String resource,
                   String role, String permission,
                   Duration duration, boolean positive)
            throws PolicyException, RemoteException
```

Works exactly as the method above, but also takes a `Duration` object that specifies when the ACL is valid. For a detailed description of durations, see 3.2.

```
public String listACL(String issuer, String resource)
              throws PolicyException, RemoteException
```

Returns the access rights of the given parameters as a string. The required permission is “read” for the specified resource path and all parent paths.

4.3.3 Interaction with PAP by Local Method Invocation

In order to communicate with PAP by local calls, the calling thread must reside in the same JVM as the PAP object. The methods signatures (excluding `RemoteException`) and semantics are exactly the same as the remote methods described above in 4.3.2. The methods are located in the class `se.kth.grid4all.security.pap.PAP`.

5 Limitations

As the current implementation of the Grid4All security infrastructure is a proof-of-concept prototype rather than of production quality, it has the following limitations that could be fixed in more matured releases.

- There is no encrypted communication; all communication is done in plain text.
- PDP can only call one VOMS, which means it can handle only one VO.
- The provided PAP client does not read the single sign-on identifier from the file system but from the command line.

6 Known Bugs

Even though PDP supports timing intervals in policies, it ignores time zones. This seams to be a problem in Sun's XACML implementation.